COMPUTER VISION ON MULTI-CORE PROCESSORS: ARTICULATED BODY TRACKING

Trista P. Chen, Dmitry Budnikov, Christopher J. Hughes, and Yen-Kuang Chen
Intel Corporation

ABSTRACT

The recent emergence of multi-core processors enables a new trend in the usage of computers. Computer vision applications, which require heavy computation and lots of bandwidth, usually cannot run in real-time. Recent multi-core processors can potentially serve the needs of such workloads. In addition, more advanced algorithms can be developed utilizing the new computation paradigm. In this paper, we study the performance of an articulated body tracker on multi-core processors. The articulated body tracking workload encapsulates most of the important aspects of a computer vision workload. It takes multiple camera inputs of a scene with a single human object, extracts useful features, and performs statistical inference to find the body pose. We show the importance of properly parallelizing the workload in order to achieve great performance: speedups of 26 on 32 cores. We conclude that: (1) data-domain parallelization is better than function-domain parallelization for computer vision applications; (2) data-domain parallelism by image regions and particles is very effective; (3) reducing serial code in edge detection brings significant performance improvements; (4) domain knowledge about low/mid/high level of vision computation is helpful in parallelizing the workload.

1. INTRODUCTION

Computer vision (CV) applications such as video surveillance, camera-assisted intelligent driving, and content-based image/video retrieval, can greatly benefit from a new computation paradigm: multi-threaded applications on multi-core processors. Unlike simple workloads such as spreadsheet calculations and media playback, CV applications are heavy in both computation and memory requirements. CV applications can benefit from multi-core acceleration of at least an order of magnitude, which is unlikely to be provided by higher clock speeds any time in the near future. Multi-core processor technology together with other architecture innovations has the potential to greatly improve CV applications' real-time performance. Furthermore, it opens the door to new algorithm and application innovations. However, the potential benefits of multi-core processors do not come for free. Via a case study, we will show readers some guidelines on how to best utilize multi-core processors.

In this paper, we study the articulated body tracking workload, which consists of the key components of a typical CV application: low and mid-level image processing from camera inputs, and high-level statistical inference. We will show that an articulated body tracking workload can achieve great parallel performance on a multi-core system, given proper parallelization. With the CV domain knowledge, we partition the work into independent tasks as follows: into image-regions in low-level preprocessing, into lists in mid-level Canny edge detection, and into particles in high-level statistical inference. Multi-threading by image regions is natural for preprocessing steps. Multi-threading by particles is also natural for statistical inference, due to the large number of particles and their independent characteristics. However, multi-threading in mid-level Canny edge detection is less straightforward. We use collections (lists) of pixels and a logically centralized queue to provide a 5x performance gain over the original implementation.

This paper is organized as follows. We first talk about typical CV techniques and applications. Detailed descriptions of the articulated body tracking application then follow. Section 3 describes the parallelization paradigm and the proposed parallelization method for Canny edge detector. Workload analysis and scalability results are shown in Section 4. We conclude the work in Section 5.

2. ARTICULATED BODY TRACKING

2.1. Computer Vision

The purpose of computer vision is to extract information about the world from images or video. Typical goals of CV include: detection, segmentation, localization, and recognition of certain objects in images; tracking an object through an image sequence; and estimation of the three-dimensional pose of humans and their limbs. Detecting/tracking objects of interest, CV is useful video surveillance, camera-assisted intelligent driving, content-based image/video retrieval, entertainment/augmented reality, robotics, etc.. This emerging field has high computation requirements, and thus can greatly benefit from the recent proliferation of multi-core processors [1].

CV algorithms can be categorized as low/mid-level image-processing and high-level statistical analysis. Image-processing techniques include various types of filtering and

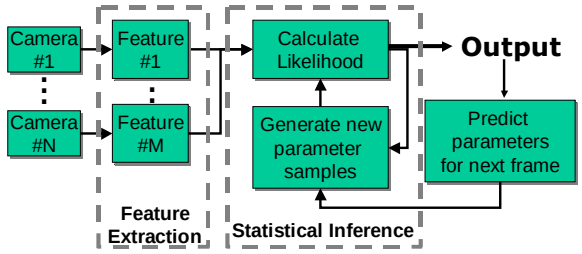

Figure 1: A general CV algorithm flow

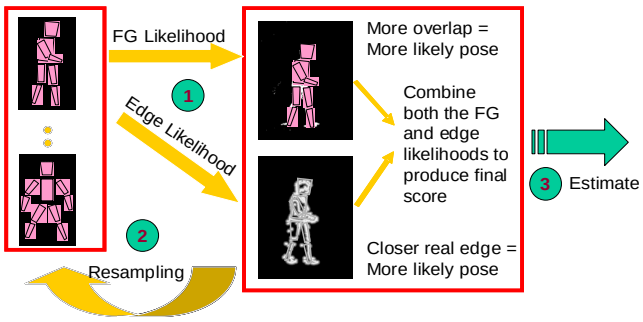

Figure 2: Articulated body tracking by annealed particle filtering

feature extraction. De-noising is an example low-level image-processing operation; Foreground detection and edge detection are examples of mid-level image-processing.

These algorithms have been extensively studied and incorporated into some well-known libraries such as OpenCV [2]. High-level analysis usually involves statistics-based approaches. Examples include the particle filtering and the EM algorithm. A general CV algorithm flow is shown in Figure 1.

2.2. Articulated Body Tracking by Annealed Particle Filtering

We adopt the method by Deutscher et al. [3] for articulated body tracking. Annealed particle filtering is a widely used method that has been shown to be robust. A kinematic-tree representation, which consists of ten body parts, is used. A pose is represented by a 30-dimensional vector describing the joint angles and locations of these ten body parts.

We have the following steps in our application:

(1) Preprocessing: This stage includes image de-noising, un-distortion and various setup operations for later stages. This is the low-level image processing stage.

(2) Foreground detection: This stage includes foreground estimation, thresholding the foreground pixels, and run-length encoding of the foreground mask. This is a mid-level feature extraction stage.

(3) Edge processing: This stage includes Canny edge detection [5], highlighting edges only inside the foreground mask, and creating a distance transform [6] for the later edge likelihood calculation. This is also a mid-level feature extraction stage.

(4) Tracking with annealed particle filtering (APF): This stage includes drawing samples of particles (each represents a pose) for likelihood calculations. The weight of a particle depends on how close the pose/particle matches the features (foreground and edge masks). Re-sampling occurs after the likelihood calculations. The process iterates until a satisfactory result is obtained. An illustration of the inference process is shown in Figure 2. This is the high-level statistical inference stage.

3. PARALLELIZATION PARADIGM

3.1. Data and Function Domain Parallelism

The two most common approaches to parallelizing applications are data-domain and function-domain. *Data-domain parallelization* partitions the data into independent pieces, each of which requires the same computation. These are operated on its unique piece of data concurrently. *Function-domain parallelization* decomposes the computation into stages, and has each thread perform one stage. In general, *data-domain* parallelization is more scalable than *function-domain* parallelization [7]. Function-domain parallelization typically suffers from significantly higher load imbalance and synchronization overhead. For example, if we use function-domain parallelization for our application: one thread would perform foreground detection, while another would perform the edge processing. However, the edge processing thread would have to wait for the foreground detection in order to highlight the edges inside the foreground mask (synchronization overhead), but also the amount of time needed to perform the two operations may differ significantly (load imbalance). Therefore, we use data-domain parallelization whenever possible.

We adopt two data partitioning schemes: *image regions* on low- and mid-level computations, and *particles* on high-level computation, as shown in Figure 3. When computing the foreground and edge features, we divide the image into many regions (image rows). Each thread will be in charge of a set of image regions. Since there are plenty of particles in the inference stage (1024), we partition by particles in that stage. Because the numbers of image regions and particles are big compared to the number of computation threads, we expect very good parallel scalability.

To parallelize the code, we use something very similar to OpenMP [4]. However the concepts we discuss and our results are not restricted to OpenMP.

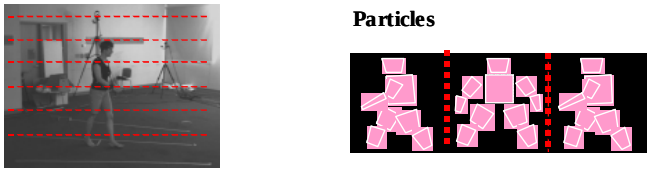

Figure 3: Data-domain parallelization by image regions & particles

3.2. Parallelizing Canny Edge Detector

While most of the modules of the articulated body tracking workload can be parallelized by a straightforward data-domain parallelization, Canny edge detector [5] does not scale well on current multi-core systems in its original implementation. It consists of (a) image gradient and edge orientation computations; (b) non-maximum suppression and high gradient pixel identification; (c) hysterisis (or "edge-growing from high gradient pixels"); (d) final assignment of the binary decision on pixels being edge or not. Steps (b) and (c) in the original Canny edge detection are done by serial operations.

We parallelize (b) by creating lists of pixels that have gradients larger than the high threshold and survive the non-maximum suppression. Each thread creates a list of pixels that it processes. After step (b), the lists are merged into a logically centralized queue. In step (c), threads grow the edge mask by operating on the pixels in the queue. Each thread dequeues one pixel at a time, checks if it is edge, and if so, enqueues its neighbor pixels. To avoid redundant work, we use an additional data structure to check whether the pixel has already been processed. If it has, we do not enqueue its neighbors. This process is analogous to a breadth first traversal of a graph. Note that the task granularity of steps (a), (b), and (d) is an image row, while that of step (c) is an image pixel. Using rows for some steps reduces the parallelization overhead. Step (c) is a per-pixel operation, and thus cannot be operated on a per-row basis. The merits of the parallelization effort will be shown in the next section.

4. WORKLOAD ANALYSIS RESULTS

We start this section by showing the parallel scalability of the articulated body tracking workload. Canny edge detector, identified as a scalability bottleneck, is then further analyzed. We show the improvement of its scalability with our method described in Section 3.2.

Figure 4. Camera inputs from four views

Table 1. Execution time breakdown and speedup of modules in 8-way SMP machine

Number of threads	*Execution time (seconds)*				*Speed-up*		
	1	*2*	*4*	*8*	*2*	*4*	*8*
Preprocessing	0.12	0.07	0.05	0.04	1.77	2.61	2.69
FG detection	0.15	0.08	0.05	0.03	1.87	3.16	4.97
Edge processing	1.48	0.86	0.54	0.62	1.73	2.73	2.40
Tracking with APF	19.28	9.67	4.91	2.50	1.99	3.93	7.71
Postprocessing	0.02	0.02	0.02	0.02	0.98	0.97	0.92
Overall	**21.06**	**10.70**	**5.57**	**3.22**	**1.97**	**3.78**	**6.54**

Table 2. Module percentage breakdown of the single-threaded articulated body tracker

Preprocessing	0.57%
FG detection	0.73%
Edge processing	7.04%
Tracking with APF	91.56%
Postprocessing	0.09%
Overall	**100.00%**

Table 3. Canny edge detector parallel speedup: original and proposed, in future multi-core machine

Number of threads	*1*	*2*	*4*	*8*	*16*	*32*
Original	1.00	1.66	2.43	3.18	3.78	4.16
Proposed	0.83	1.67	3.25	6.28	11.92	19.91

The test sequence is shown below with four camera views. The resolution of each image is 640x480. We first analyze the full application on an 8-way symmetric multiprocessor (SMP) machine. The machine has eight Intel® Xeon™ processors. More detailed analysis is done on a simulated future multi-core system. In the paper, the number of threads is the same as the number of cores in all the experiments. That is, we use one thread per core.

Seven frames (plus twenty warm-up frames) of video are used in our SMP experiments. Table 1 shows us the execution time of each module in seconds. One, two, four, and eight thread results are shown. The speedup of the eight-thread system is very good at 6.54 (Table 1). Table 2 shows that tracking with APF takes up over 91% of the execution time for a single threaded run. This stage scales extremely well: 7.71 out of 8 threads.

The results in Table 1 show that edge processing will become a significant bottleneck as we scale to even more cores. Although it takes only 7% of the total time for a single-threaded run, its scalability is much worse than tracking with APF. Thus, we further analyze this stage on a simulated future multi-core system.

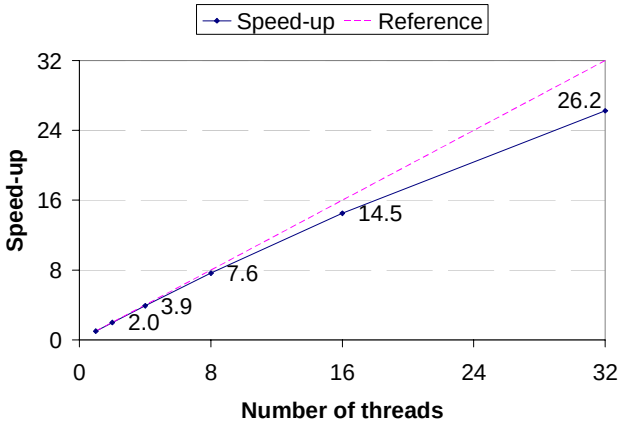

Figure 5. Parallel scalability of articulated body tracker on a future multi-core machine. The reference line shows linear scaling.

Table 4. Percentage of time on synchronization, serial code, and load imbalance on a future multi-core machine

Number of threads	*1*	*2*	*4*	*8*	*16*	*32*
Synch+serial+imbalance (%)	0.01	0.63	1.81	4.12	8.62	16.75

4.1. Canny Edge Detector Parallelization Results

Canny edge detector, as described in Section 3.2, has serial operations in the original implementation that prevents it from performing well in a multi-core system. We compare the original version of this module with our proposed implementation on the simulated future multi-core system. Our proposed method gives a 19.9/32-core speedup compared to 4.16/32-core of the original implementation. This is almost a 5x performance gain.

4.2. Scalability of Articulated Body Tracking Workload

Figure 5 shows the parallel scaling of the optimized articulated body tracking workload on a future multi-core system. The workload scales well, giving a 26.2/32-core speedup. The remaining non-linearity in the speedup curve comes from synchronization overhead, serial code, and load imbalance as shown in Table 4.

5. CONCLUSIONS

Looking forward, we believe the number of cores per processor will increase [8]. This is because parallel processing is a power-efficient way to increase performance. Computer vision applications have the potential to run much faster by utilizing ever more powerful multi-core systems.

On the other hand, algorithms and applications must be tuned to allow multi-core processors to exploit their inherent parallelism. We have shown the importance of properly parallelizing a workload in order to achieve great performance. Our study has shown that: (1) data-domain parallelization is a good fit for CV applications and avoids problems inherent to function-domain parallelization; (2) data-domain parallelization by image regions and particles is very effective; (3) the proposed Canny edge detector scales well, with a 5x performance gain compared to the original implementation; (4) our CV domain knowledge helped us in choosing image regions and particles for data-domain parallelization and helps us in improving Canny edge detection.

Understanding distinct characteristics of low/mid/high-level CV operations helps us deliver scalable multi-threaded CV applications that best utilize the emerging multi-core computation paradigm. For example, in the Tracking with APF module, we did not partition the work by image regions because of two reasons: (1) partitioning by particles results in more tasks, and (2) particles are independent of each other. Therefore, in this high-level CV operation, we divide the work in terms of particles.

In the future, instead of minor algorithm optimizations, we should concentrate on new algorithm research targeting the exploitation of thread-level parallelism. While low-level CV algorithms are often easily parallelized, we must pay more attention to mid-level CV algorithms and high-level CV algorithms. Future algorithm research should keep in mind that future platforms will have many cores so that the algorithms will perform well on those platforms.

6. ACKNOWLEDGEMENTS

The authors thank Sanjeev Kumar, Roman Belenov, Konstantin Rodyushkin, and Horst Haussecker from Intel for insights into this workload; and Prof. Michel Black and Alexandru Balan from Brown University for suggestions on the algorithm.

7. REFERENCES

[1] T. P. Chen, H. Haussecker, A. Bovyrin, R. Belenov, K. Rodyushkin, A. Kuranov, and V. Eruhimov, “Computer Vision Workload Analysis: Case Study of Video Surveillance Systems”, *Intel Technology Journal*, 9(12), May 2005.

[2] Open Source Computer Vision Library (OpenCV): http://www.intel.com/research/mrl/research/opencv.

[3] J. Deutscher, A. Blake and Ian Reid, “Articulated Body Motion Capture by Annealed Particle Filtering”, *CVPR 2000*.

[4] Open SMP Programming (OpenMP): http://www.openmp.org.

[5] J. Canny, “A Computational Approach to Edge Detection”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol 8, No. 6, Nov 1986.

[6] Donald G Bailey, “An Efficient Euclidean Distance Transform”, *10th International Combinatorial Image Analysis Workshop (IWCIA 2004)*, Auckland, New Zealand, December 1-3, 2004, pp. 394-408.

[7] Parallel Computer Architecture: A Hardware/Software Approach. D. E. Culler, J. Pal Singh, and A. Gupta. Morgan Kaufmann Publishers 1999.

[8] W. W. Gibbs, "A Split at the Core", Scientific American, pp. 96-101, Nov. 2004.